Replication and Archiving - User's Guide

Table of contents

1. Introduction 3
2. Concepts and terminology 4
   2.1. Content package 4
   2.2. Original server/vault 4
   2.3. Full replica server/vault 5
   2.4. Cached replica server/vault 5
   2.5. Delta export 5
   2.6. Shortcut objects 5
   2.7. Dropout objects 6
   2.8. Conflict objects 7
   2.9. Remote checkout 8
   2.10. Versioning changes 8
   2.11. Non-versioning changes 8
3. Use cases 9
   3.1. Two-directional full content replication 9
   3.2. Full and optimized access to critical data, generic access to everything else 12
   3.3. Replication to maintain a warm standby between on-premise and cloud 13
   3.4. Replication in a chained network topology 14
   3.5. Replication to improve the observed performance of a cloud vault 15
   3.6. Publishing to another vault 15
   3.7. Creating a long-term archive 17
   3.8. Transferring content packages between servers 18
   3.9. Importing specific objects only 18
4. Key features 20
   4.1. Mapping different metadata structures between vaults 20
   4.2. Object identification across multiple vaults 20
   4.3. Mapping object identities across multiple vaults 21
   4.4. Detecting conflicting changes across vaults 22
   4.5. Resolving conflicts 23
5. Frequently asked questions 24
   5.1. Are there any additional requirements for a successful mapping of metadata structures? 24
   5.2. Does unmappable metadata survive replication roundtrips? 24
   5.3. My replica server has a set of users that is completely different from that on the original server. How do I map the users and permissions in this case? 24
   5.4. Do M-Files URLs work across multiple vaults? 25
   5.5. Do embedded file links work across multiple vaults? 26
   5.6. My new value list item appears as "My New Item (deleted)" in other vaults, why? 26
   5.7. How do I find objects that have property values referring to deleted items? 26
   5.8. Is it possible to modify the metadata structure once the replication has started? 26
   5.9. I receive e-mail notifications from multiple vaults for the same object, why? 27

5.10. What happens if the local object has newer changes than those being imported from a content package? 27
5.11. Some objects in the original vault are not exported, why? 27
5.12. Some objects in the content package are not imported, why? 27
5.13. Some object data in the content package is not imported, why? 28
5.14. When an import job is run in my vault, the timestamps indicating the latest run of export jobs seem to be adjusted backwards. Why? 28
5.15. I have configured my export job to save files as PDF/A. Why cannot I find them in any remote vault after import? 29
5.16. Is it possible to drop IDs from the names of the files in the content package? 29
5.17. Non-admin users are not allowed to use the "Mark for archiving" command any longer. How do I make that possible again? 29
5.18. What exactly is stored into a conflict object? 29
5.19. What am I supposed to do with conflict objects? 29
5.20. I am replicating my object between two vaults, and now I have a conflict object at both ends. How do I resolve this conflict correctly? 30
5.21. What are the timestamp properties relevant to this feature? 30
5.22. Can I use metadata-driven permissions normally via shortcut objects? 31
5.23. Can I use any event handlers with replication and archiving? 31
Appendix A. Common registry settings 32
Export 32
Import 35
URL handling 37
Appendix C. XML processing 38
Using XSL transformation to select which objects to import 38

1. Introduction

This document provides additional technical information on the Content Replication and Archiving feature in M-Files. M-Files User's Guide describes the basic functionality, but the viewpoint here is more detailed and solution-oriented. The primary purpose of this document is to give M-Files Administrators, Consultants, and Support Engineers some further ideas of using and troubleshooting the feature.

In M-Files, there are two mechanisms available for replication: full replication and cached replication. Full replication is based on content packages that are exported and imported between original and replica servers, replicating both the metadata content and the file data. Cached replication is based on an intermediate server that relays the network traffic from clients to the original server, but additionally caches the file data in the process as a read optimization.

For the rest of this document, "replication" always means full replication unless mentioned otherwise.

2. Concepts and terminology

2.1. Content package

A content package wraps all the exported data for a single content export. Content packages have an open format and include the following files and directories:

- Index.xml
    - Content package header file containing common information and statistics about the content package
    - Includes XML Entity references to other related XML files of the content package
- Ready
    - Semaphore file indicating that the content package is ready for import
    - In multi-import mode (importing all the available content packages from a specific directory, as opposed to importing a specific content package), the package is imported if "Ready" file exists and "Imported.<vaultguid>" does NOT exist for the importing vault
- Imported.<vaultguid>
    - Semaphore file indicating that this content package has been imported successfully to a vault identified by <vaultguid>
    - In multi-import mode (importing all the available content packages from a specific directory, as opposed to importing a specific content package), the package is imported if "Ready" file exists and "Imported.<vaultguid>" does NOT exist for the importing vault
- Files
    - Folder sub-tree containing the file data
    - Folder structure has the same structure as the ID2 view in M-Files
- Metadata\Structure.xml
    - XML file describing the metadata structure elements used for the exported object metadata
    - Metadata structure mapping is based on the information available in "Structure.xml"
- Metadata\Content<nnn>.xml
    - XML files containing the metadata for exported objects
    - Each "Content<nnn>.xml" represents one batch of objects (batch size can be adjusted by registry settings)

2.2. Original server/vault

A source server (and vault) from which content packages are exported.

2.3. Full replica server/vault

A target server (and vault) to which content packages are imported from the original server/vault. The original vault and the replica vault always have different identities.

There is no need to have a continuously online network connection between the original server and the replica server. Network connectivity is only required when a content package is transferred from the original server to the replica server.

The roles of originals and replicas are bound to one-directional replication, and therefore reversible in two-directional replication.

2.4. Cached replica server/vault

An intermediate server (one that is geographically closer to the users) that relays the network traffic from clients to the original server. The cached replica vault caches the file data as a read optimization, but forwards other client-server communication all the way to the master server. Therefore, continuous online connectivity is required for using a cached replica server.

The original vault and the cached replica vault have the same identity (vault GUID).

2.5. Delta export

In delta export, only the changes since the specified delta timestamp value are exported. Typically, the delta timestamp is derived from the latest run of the content export job. In some use cases, however, the delta timestamp can be dynamically configured to reflect the change timestamps of recently imported changes. This allows the imported changes to be accounted for the same relevance with the local changes when replicating changes in a chained network topology.

There is no separation between changes in metadata or file data. Therefore, if a new version has been created due to metadata changes only, the unchanged file data is still exported even if using delta export. This is a probable area of improvement in future versions of M-Files.

2.6. Shortcut objects

A shortcut object is a link object that is used as a placeholder when having relationships to objects in a remote vault. Each shortcut object has the same global identity as its remote counterpart it represents, but no actual operations can be carried out on the remote object via a shortcut object. Shortcut objects simply provide a means of navigating along the cross-vault relationships.

Shortcut objects can be created in the following ways:

- Creating manually a relationship from a local object to a remote object. For instance, an object from vault B is dragged and dropped as a relationship to another object in vault A.
- Replicating objects that have a relationship to another object that is not replicated (and does not exist in the replica vault)

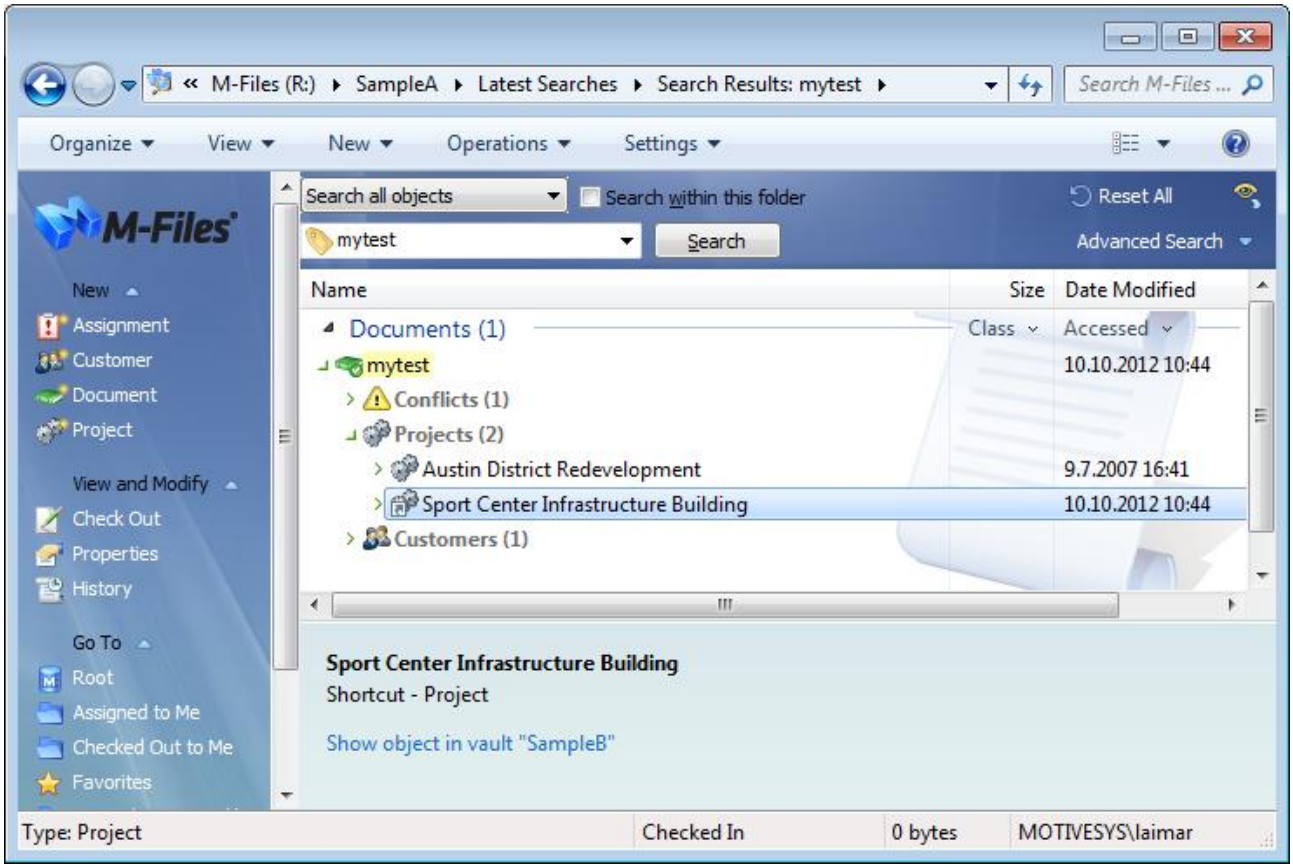

Fig. 1. Shortcut object shown as a related object.

2.7. Dropout objects

M-Files Server remembers the exported objects within the context of a specific scheduled content export job. When the same job is run again, it is possible to export "dropout" information on objects that are no longer part of the exported data. In other words, so called dropout objects have been exported at least once by a specific export job, but they are not any longer exported by the latest run of that same job. Usually the reason for this is that the object does not match the export filter any longer.

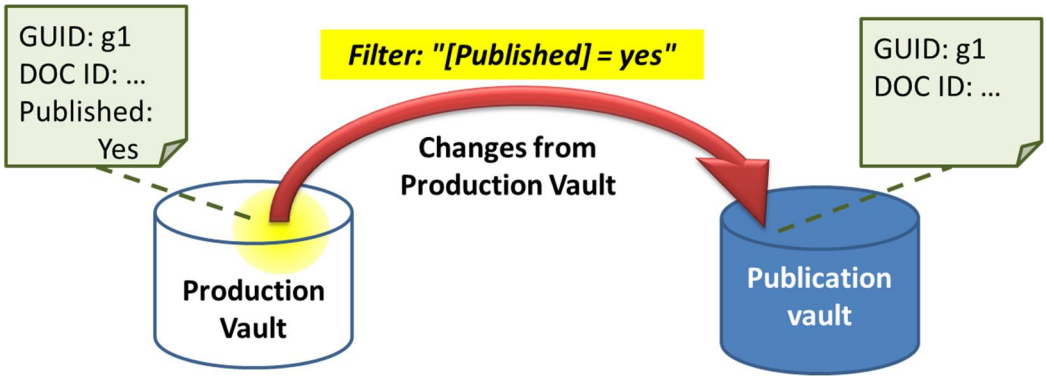

Fig. 2. Phase A: replicating an object that matches the export criteria.

On the import side, dropout objects cause the replicated object to be destroyed from the replica vault. The object incarnation in its original vault, however, is never destroyed because of dropouts. In other cases, to

protect oneself against accidental dropout objects and their destructive effect on local objects, content import jobs can be configured to disallow any object destructions.

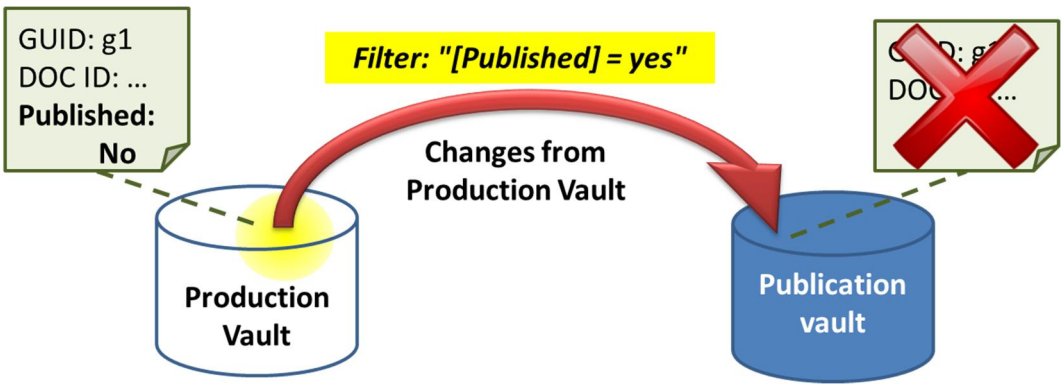

Fig. 3. Phase B: replicated objects are destroyed when they do not match the export criteria any longer.

For every content export job, the exported objects are logged into the database, thus providing a simple audit trail on exports if needed. However, the actual dropout information is available for scheduled content export jobs only, and not for temporary content export jobs that are run only once.

2.8. Conflict objects

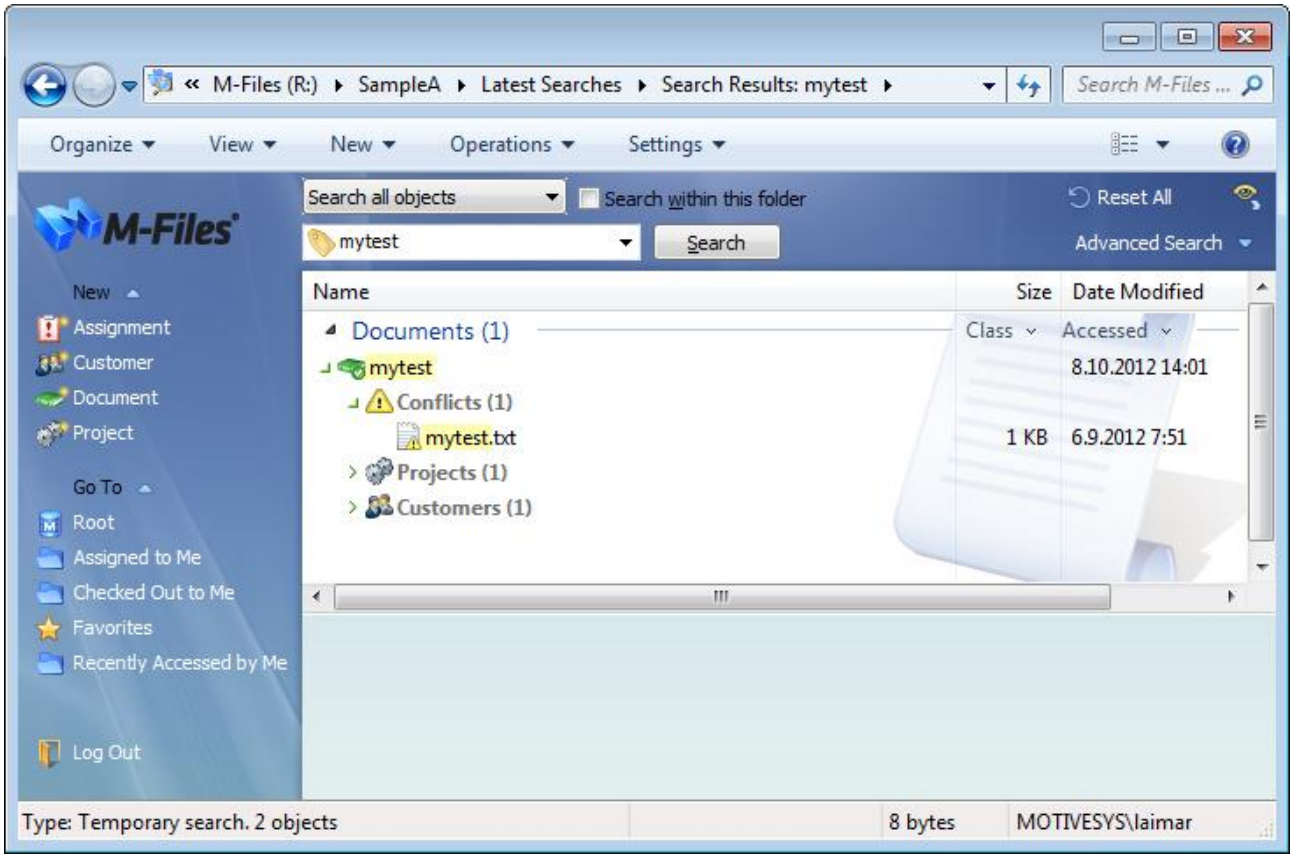

Fig. 4. Conflict object as a related object.

When replicating changes from one vault to another, there may exist new object versions that have been independently created in both vaults. Each object version has a globally unique ID in addition to the version number, and if there is a mismatch when comparing those two between the local data and the imported

data, a conflict is detected. Such conflicting changes are not imported normally, but a specific conflict object is created from the imported data (i.e., conflicting versions).

A conflict object contains all the incoming object versions that conflict with the local data. Additionally, each conflict object has a relationship to its local counterpart, which makes them accessible via related objects hierarchy in view listings. Conflict objects are also listed in the built-in view "Conflicts" (hidden by default).

Conflicts can be resolved by keeping either the local changes or the conflicting changes.

2.9. Remote checkout

In order to mitigate the risk for conflicting modifications across multiple vaults, remote checkout is used to represent an outstanding checkout status in another vault. When observed by the users, remote checkout is very similar to local checkouts but with no checked-out version available. Also, remote checkout should be interpreted as object-level information only, because the version history may not match that of the original object.

Remote checkout information is exported with the normal data replication, and it is imported unless the content import job explicitly prohibits the checkout state to be imported. Note that the checked-out version is NOT included in the content package. Additionally, a remote checkout never overrides a local checkout.

If the user has sufficient access rights, a remote checkout can be undone similarly to normal checkouts.

2.10. Versioning changes

Versioning changes are modifications that create new versions on the modified object. File data modifications and most metadata modifications are versioning changes.

When importing versioning changes, the version identities are compared between the local data and the incoming data. If there is a mismatch between the version identities, a strong conflict is detected and a separate conflict object is created (see "2.8 Conflict objects").

2.11. Non-versioning changes

Non-versioning changes are modifications on objects that can be made without creating a new version: modifying permissions, modifying a version label, modifying a version comment, deleting an object, or undeleting an object.

Typically, when importing non-versioning changes, incoming changes that are older than the local changes are ignored because a weak conflict is detected. In many cases, such weak conflicts are reported to Windows Event Log.

3. Use cases

3.1. Two-directional full content replication

This use case illustrates how to start replicating all the data between two geographically distributed sites. In this case, the main driver to use replication is the poor performance of wide area network connections between the remote sites. When using the M-Files client to access the replicated data, the performance is optimal within the local area network. As an additional benefit, client access is uninterrupted even if the network connection between the sites is occasionally offline.

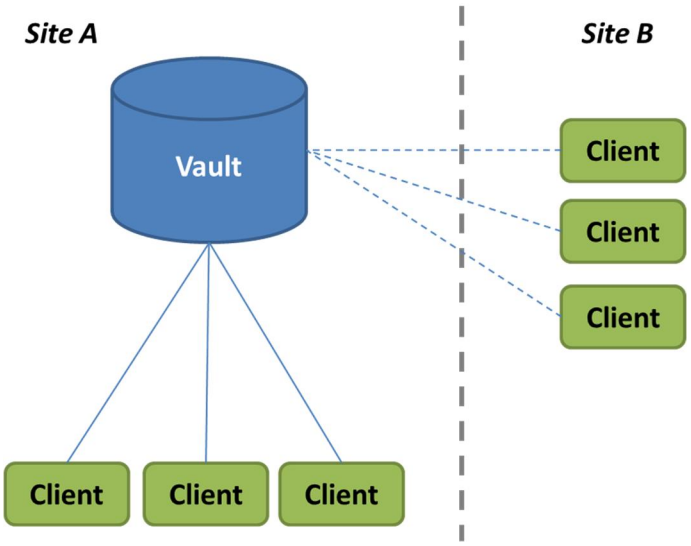

Fig. 5. Multi-site access without content replication.

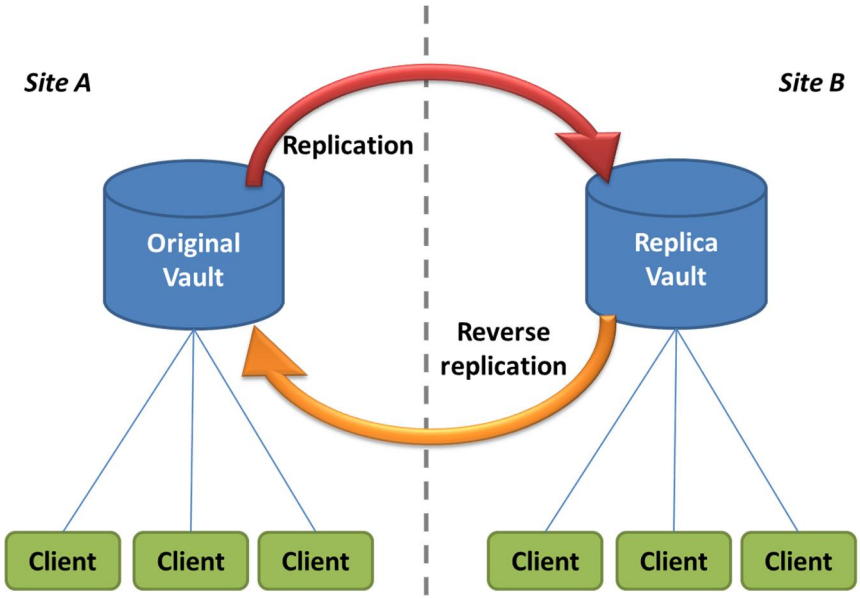

Fig. 6. Multi-site access with content replication.

1. Install an appropriate license to the replica server. Note that you are not allowed to use the same license code on your original server and replica server. You must obtain a license code with a different serial number for installation on the replica server. Each M-Files Server must have a

license code with a unique serial number. If unsure, please contact sales@m-files.com to review your current license status.

2. Copy a recent full backup of the vault to the replica server.

3. Restore and/or attach the vault backup to M-Files Server on the replica server **with a new identity**. Allow M-Files to initially **disable the features that may cause external side effects**. Also note that all **checkouts are automatically undone** when the vault is attached with a new identity.

4. Enable login accounts on the replica server as appropriate. Login accounts have automatically been created in disabled state for all user accounts in the attached vault. Verify that the license type of the login is appropriate. You will typically only want to enable those users' login accounts that need to access the vault on the replica server. Your M-Files license for the replica server should include a sufficient number of named user licenses to cover the number of users that access the vault on this replica server.

5. Do not make changes to Users and User Groups. Typically, it is desirable that any Active Directory integration remains active in the replica vault, too, i.e., user accounts and user groups continue to be synchronized from the organization's Active Directory into both the original vault and the replica vault. Internal IDs of new user accounts and user groups will vary between the vaults, but the login account name in Active Directory acts as the identifier for user accounts, thus maintaining a unified identification between the vaults for these entities.

6. Consider which external object types and value lists should be left as external (assuming that the databases are reachable from the replica server), and either enable them as external object types or make them internal object types in M-Files Server Administrator. It is typically best that each vault instance synchronizes objects such as customers and contact persons from the master CRM database independently. The External IDs of these objects act as the identifiers, maintaining a unified identification between the vaults for these entities even if the internal IDs of the objects will vary between the vaults. Note that external objects are not currently imported at all, thus any modifications on custom properties (i.e., those properties that are not part of the database synchronization) are not replicated.

7. Consider which connections to external file sources and mail sources are relevant in the replica vault, and enable them in M-Files Server Administrator. Typically, it is desirable that only the original vault instance retrieves new documents from such sources, and that they then get replicated to other vault instances by M-Files. However, if the new office has a scanning station of its own, it makes sense to keep the import job for that scanner in the replica vault, and delete the duplicate job from the original vault.

8. Leave all Reporting and Metadata Export jobs disabled in the replica vault. It is desirable to export the same data sets from the original vault only.

9. Consider the need for backups locally for each replica vault. If the full content is replicated, backing up the master vault only is the minimum level of a disaster recovery plan. However, if a faster disaster recovery is needed for replica vaults, it is useful to have local backups for them as well.

10. Consider workflows. Specifically, consider whether the definitions and actions in each workflow are appropriate for a replica vault. Typically workflows need not be changed.

11. Consider e-mail notifications. Currently, e-mail notifications are processed independently in each vault participating in replication, thus potentially sending duplicate notifications for a single object. The simplest workaround is to have each user defined in her original vault only, unless the users

need roaming access to multiple locations. Another possibility is to keep each user's notification rules only in her original vault, and remove the rules from replica vaults.

12. Set up the full content export jobs in M-Files Server:

    a. On the original server, create a scheduled content export job with no filter. Turn on the delta option ("Export only changes since latest run") and specify a timestamp that is at least a couple of hours older than the timestamp of the database backup that was used for restoring the vault's database on the replica server (if unsure, add a couple of days as a safety margin by specifying an older date). This is necessary to ensure that all content that has changed after the backup was created will also be replicated to the replica server.

       For the target location, specify a network file share that is accessible by the replica server (e.g., you may want to share the folder "C:\M-Files Content Packages\<vault>\Out" on the original server). If a commonly accessible network folder is not available, you need to manually take care of transferring the content packages between the servers. For example, you may set up a Windows Task Scheduler job that copies the content packages by using FTP or some other suitable transport. If you have an inquiry on our solution using an FTP transfer tool, please contact sales@m-files.com.

    b. Start the job.

    c. On the replica server, create a scheduled content import job, pointing to the folder where the export job is producing the content packages.

       Once the above mentioned export job has completed, start the import job.

    d. On the replica server, create a scheduled content export job with no filter. Turn on the delta option and specify a timestamp that is at least a couple of hours older than the time at which the vault was attached to the replica server. This is necessary to ensure that all content that has changed in the replica vault after the vault was attached/restored will get replicated back to the original vault. Even if there has been no user activity, some objects may have been created or changed because of connections to external databases, for example.

       Once the above mentioned import job has finished, start this export job.

    e. On the original server, create a scheduled content import job, pointing to the folder where the export job is producing the content packages on the replica server.

       Once the above mentioned export job has completed, start the import job.

13. Consider setting up the content export jobs also for destroyed data in M-Files Server:

    a. On the original server, create a scheduled content export job with the option for destroyed objects and object versions turned on. Otherwise, follow the instructions in 12.a.

    b. On the replica server, follow the instructions in 12.c. Ensure that the "Do not import object destructions" option is turned off for the content import job.

    c. On the replica server, create a scheduled content export job with the option for destroyed objects and object versions turned on. Otherwise, follow the instructions in 12.d.

d. On the original server, follow the instructions in 12.e. Ensure that the "Do not import object destructions" option is turned off for the content import job.

14. Create or import the necessary login accounts on the replica server. Typically, you will want to import users from Active Directory in order to create login accounts for those users who will be accessing the replica (e.g., users at a specific site). Ensure that the login accounts of those users exist on the replica server, are enabled, have the correct full name and e-mail address information, and have a valid license type assigned to them. Also, verify that the user account in the vault is enabled (should be enabled by default).

3.2. Full and optimized access to critical data, generic access to everything else

This use case illustrates a more optimized way of using two-directional replication between remote sites. Instead of replicating all the data, it might be more efficient to replicate only the critical data, and access other "non-critical" data from the master vault only. This approach is an optimization on both the network connectivity and the network bandwidth usage: frequently used data (but not everything) is replicated to a vault closer to the users, while more rarely used data is available from the master vault whenever the network connections are decent.

For example, a remote project site could be implemented by this use case. The project site would have a replica vault containing all the documents and additional data related to the project, but the rest of the company documents and other data would remain on the original server.

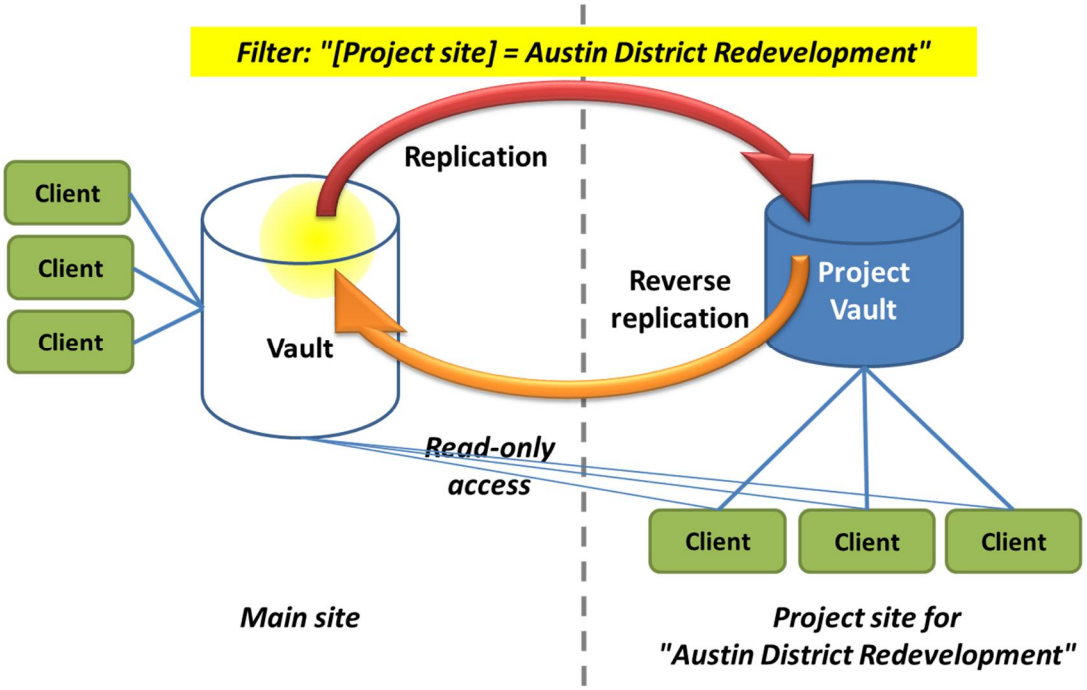

Fig. 7. Replication between the main site and the project site.

1. Replicate the critical data by following the instructions in "Two-directional full content replication". For steps 12.a and 12.c, instead of using no filter with content export jobs, use a specific filter that captures the critical data.

    a. Beware that pseudo-user references in object permissions may not work as originally intended unless all the involved objects are replicated. This is due to the fact that pseudo-user references are resolved based on the local data only. Therefore, it is important to

carefully design what critical data really means, because the resulting permissions depend on what data is actually replicated.

b. You may need multiple content export jobs to support filtering all the critical data. It does not matter if one object is exported by multiple jobs: if the exported content is the same in multiple content packages, only the first imported package will effectively import the possible changes to the replica vault. Another option is to use additional properties to indicate relevant objects from various object types.

2. Configure the clients to use two vault connections, one for the local replica vault and another for the original vault (possibly via a cached replica server). By using this configuration, the critical data is always available from the local vault. At the same time, however, it is possible to access all the other data if a network connection is available.

   a. To avoid unnecessary parallel modifications, consider assigning read-only licenses in the master vault for users that primarily use their local replica vault.

3.3. Replication to maintain a warm standby between on-premise and cloud

This use case illustrates a hybrid approach that easily combines high availability with high performance on-premise deployments, and uses the cloud server as a backup for an on-premise vault. Users normally access the on-premise server (original server) from which all the master data is replicated to a cloud server (replica server). If the on-premise server becomes unavailable (due to a severe hardware failure, for instance), it is relatively easy and quick to switch over to the cloud server and continue using the vault. Because the master data is located in the the on-premise vault, the cloud vault(s) are discardable in this scenario.

If the administrator wants to eliminate the possibility of conflicting modifications in the cloud server vault while the on-premise server and vault are being recovered, read-only licenses can be assigned to the users of the cloud server vault.

In order to complete all the steps of this use case, some preparation and support is needed from M-Files Cloud Vault administrators.

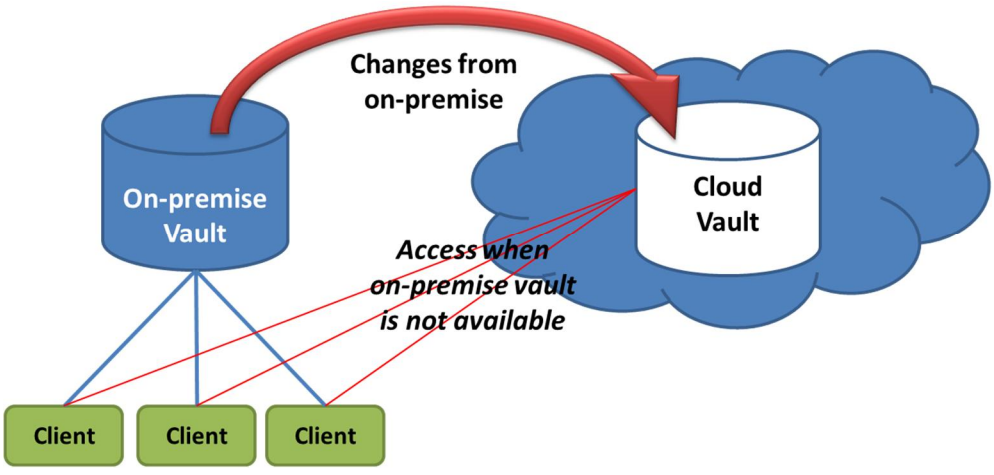

Fig. 8. Replicating and accessing a warm standby in cloud.

1. Follow the steps 1-11 of the instructions in "Two-directional full content replication". In this use case, the replica server is located in Windows Azure.

2. Install Windows Azure Connect on the on-premise server. Details on how to do this are available from the Windows Azure Connect pages on the Microsoft MSDN site: (http://msdn.microsoft.com/en-us/library/windowsazure/gg433122.aspx)

3. In order to make the servers aware of each other, add both the on-premise server and the Windows Azure server to the same group in Windows Azure Connect.

4. Configure a shared folder for transferring content packages between the on-premise server and the Windows Azure server. The following example shows how to do this: http://blogs.microsoft.co.il/blogs/applisec/archive/2011/01/22/how-to-enable-file-share-with-azure-vm.aspx

    a. Note that the Windows Azure end of this configuration must be part of the instance startup script in the Azure package.

5. Follow the steps 12-13 of the instructions in "Two-directional full content replication" to set up the content export and import jobs. For the content package location, use the shared folder configured in the step 4 above.

6. In case of an on-premise failure, a quick failover can be accomplished simply by reconfiguring the client vault connections to use the Windows Azure.

    a. Note that because the vaults have different identities, checked out objects cannot be recovered after the failover.

3.4. Replication in a chained network topology

This use case illustrates how to forward replicated changes between multiple vaults on a hop by hop basis, without the need of full point-to-point replication. A daisy chain is the simplest example of a chained network topology, replicating changes from vault A to vault B, and further from vault B to vault C. A hub-and-spokes topology is similar to daisy chain, except that there is only a single forwarding node: modifications made at non-hub nodes are replicated to other non-hub nodes via the hub node.

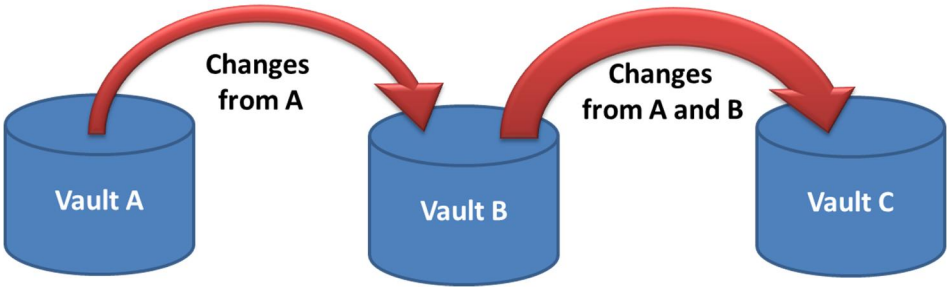

Fig. 9. Replicating changes from A to C via B.

1. Follow the steps 1-11 of the instructions in "Two-directional full content replication" to set up all replica servers and vaults.

2. Set up the full content export jobs in M-Files Server:

    a. On each original server, follow the steps 12.a and 12.d of the instructions in "Two-directional full content replication".

    b. On each replica server, follow the steps 12.c and 12.e of the instructions in "Two-directional full content replication". To allow the changes to be further replicated

whenever imported to an intermediate replica vault, turn on the "Reset export timestamps upon import" option of the content import jobs.

3. If you also want to replicate destroyed objects and object versions, apply the step 2 above to the step 13 of the instructions in "Two-directional full content replication".

3.5. Replication to improve the observed performance of a cloud vault

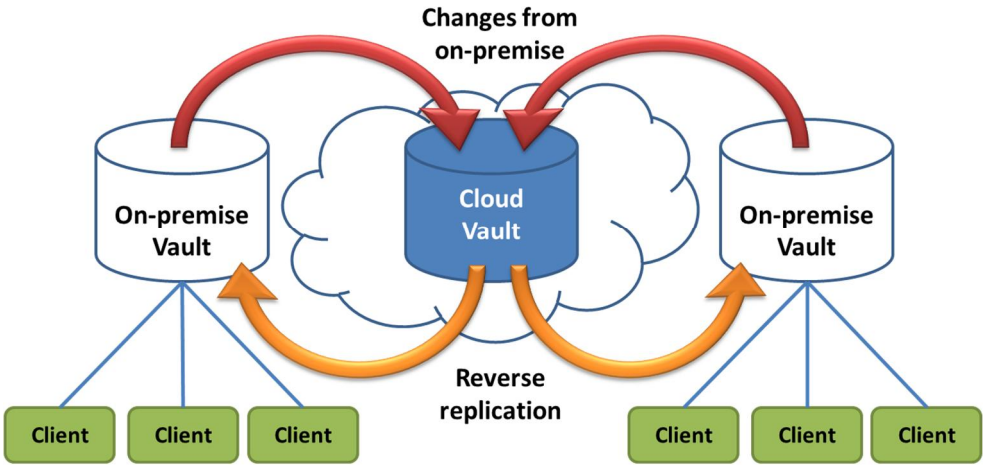

Fig. 10. Using on-premise vaults as front-ends for a cloud vault.

This use case is technically the same as the use case "Replication to maintain a warm standby between on-premise and cloud" described above. However, here the roles of original and replica servers are reversed: the cloud-based server is the original server and the on-premise server is the replica server. In practice, this role reversal affects the notion of master data and how different nodes are deployed in the system. This use case deploys the master data in the cloud vault, and potentially multiple on-premise nodes can be deployed as replica servers (discardable when not needed any longer). On-premise vaults typically provide slightly better performance than cloud vaults; therefore using this deployment will often improve the end user experience.

If the deployment comprises more than one on-premise vault, the changes from them should be replicated back to each other by applying the "Replication in a chained network topology" use case to the reverse replication from the cloud vault. Note that in the current implementation, the changes originating from the on-premise vault A are sent back to the vault A again. Such an attempt to import already existing changes does not cause any harm in the target vault, but it will increase the amount of data being transferred.

3.6. Publishing to another vault

This use case illustrates how to use one-directional replication to make selected objects available via a publishing vault. Replicated data is typically read-only in the publishing vault, so there is usually no need to replicate changes back to the original vault.

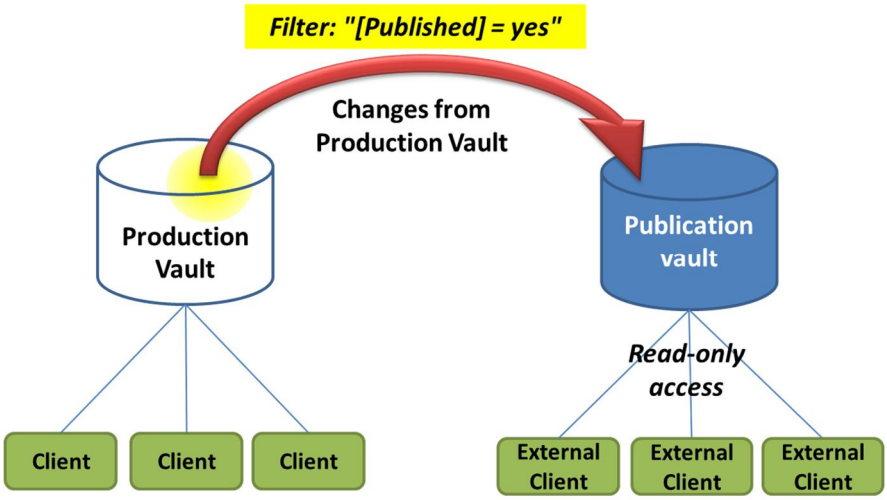

Fig. 11. Replicating to a publication vault.

1. Follow the instructions of the step 1 in "Two-directional full content replication" in order to prepare a replica server for publishing.

2. Create a replica vault on the replica server.

    a. If you want to utilize the metadata structure of the original vault for the publishing vault, you can also use the original vault as a starting point by following the steps 2-3 of the instructions in "Two-directional full content replication".

3. Configure the metadata structure of the replica vault suitable for publishing.

    a. Add semantic aliases to both the original vault and the replica vault to map the metadata structure between them.

    b. For properties based on the Users list, you may want to specify fixed values in some cases. This is especially the case when the sets of users are completely different between the vaults, and you want to avoid deleted User items to be created upon content import. For more details, see the registry setting "ArchiveRestorePropertiesWithSpecificUserID".

    c. You may want to specify the property definitions for which property values are skipped when importing the objects. For more details, see the registry setting "ArchiveExcludePropertiesUponRestore".

4. Create and enable login accounts on the replica server as appropriate. Typically, the login accounts on the publishing server are configured as external. Verify that the license type of each login account is appropriate (typically read-only). Your M-Files license for the publishing server should include a sufficient number of read-only user licenses to cover the number of users that access the vault on this replica server.

5. Consider the need for backups locally for the publication vault. If the publication vault is backed up locally, disaster recovery is potentially faster than recovering all the way from the original vault.

6. Set up the content export jobs in M-Files Server:

    a. On the original server, create a scheduled content export job with a suitable filter that captures the objects to be published. As a simple approach, this can be accomplished by using a boolean property "Published" that controls which objects are to be published. Additionally, turn on the delta option to allow only the changes to be replicated from the

original server to the publishing server, and initially specify a delta timestamp that is older than any existing object in the vault.

For the target location, follow the instructions of the step 12.a in "Two-directional full content replication".

b. On the publishing server, create a scheduled content import job, pointing to the folder where the export job is producing the content packages. Consider using the "Latest version only" option for limiting the amount of data to be transferred to the publication vault.

Once the above mentioned export job has completed, start the import job.

7. For publication use, it is not usually needed to replicate the destroyed objects and object versions, but you probably want to replicate the dropout objects.

   a. Include the dropout information in exported content packages by turning on the dropout option ("If a previously exported object is no longer part of the export set, mark it to be destroyed in the target vault") on the content export job.

   b. Note that the dropout information is applied only within a single scheduled content export job. Multiple jobs do not co-operate with each other: each job is responsible for its own dropouts only, and objects continuously exported by one job may end up as dropouts by another job.

   c. Note that changing the export filter of an existing content export job may have undesired side effects when combined with dropout information. This is mainly because, by definition, the objects may no longer be part of the export set. Therefore, if the export filter is radically changed, it is often better to start using a new export content job with it.

3.7. Creating a long-term archive

This use case illustrates how to use replication and archiving to create a long-term archive. The basic requirement of long-term archiving is to store the data in a format that is independent from the applications that originally produced the data. In M-Files, the content packages fulfill this requirement by using widely accepted XML for the metadata content and normal files and folders for the file data. For additional independency from application-specific file formats, the PDF/A format can be used as a long-term representation for most Microsoft Office document formats. As a result, the contents of the long-term archive are accessible and usable even if the M-Files software is not available.

1. Set up the content export job in M-Files Server:

   a. Create a scheduled content export job with a suitable filter that captures the objects to be archived. The filter might be something like "employment agreements that have expired more than one year ago". The job can be scheduled to run once a month, for example. In many cases, the archiving jobs typically have much more relaxed scheduling requirements than those of continuous replication.

   b. In order to archive documents in an acceptable format for long-term archiving, turn on the PDF/A-1b option for the content export job. Note that the PDF/A files are generated **in addition to** the original files, and links to them are provided in the content XML file of the content package. Also note that currently the PDF/A files are only generated for most Microsoft Office document formats and PDF documents that are not yet in PDF/A format.

c. Note that archiving old versions is not currently available as a scheduled content export job. Old versions can be archived, however, by using a context menu command available from the "Content Replication and Archiving" node in M-Files Server Administrator's tree view.

2. Manually transfer the archived content packages to some permanent media, or set up a separate Windows Task Scheduler job to perform this task automatically.

3. Once archived, the content packages can also be imported to another "archive" vault to provide more convenient searching and browsing.

3.8. Transferring content packages between servers

The current out-of-the-box implementation of replication and archiving does not provide any means of transferring the replicated data, but it simply relies on shared folders visible to both ends. If such a shared folder is not sufficient to a particular replication solution, other transfer mechanism can be used as required. For example, one relatively simple mechanism is to use FTP to transfer zipped content packages from the original server to the replica server. If you have an inquiry on our solution using an FTP transfer tool, please contact sales@m-files.com.

In order to properly transfer the content packages from one server to another, consider the following aspects:

- On the original server, ensure that all the available packages are transferred in their chronological order.

- Ensure that each content package is successfully transferred and imported on the replica side. Should some packages be skipped, not all the data will be available in the replica vault.

- Before the transfer, content packages are zipped without the "Ready" indicator file. When the zip package is unzipped on the replica side, the content package import is not triggered too early by accident.

- The zip package is split into smaller chunks on the original server before the FTP transfer. This ensures that even large content packages can be transferred over unreliable network connections. Once all the chunks have been transferred successfully, they are gathered together as a complete zip file on the replica side.

3.9. Importing specific objects only

This use case illustrates how to import individual objects from a content package.

1. Make the content package available **without the "Ready" file** in the location from which the content import jobs can later access it.

2. Use XSLT transformation on "Content<nnn>.xml" files to filter out any unwanted objects and to re-generate "Content<nnn>.xml" files to contain only those objects that are to be imported. An example on how to do this can be found in "Appendix C. XML processing".

    a. The actual set of imported objects is determined by the content of the "Content<nnn>.xml" files. In other words, it does not really matter if there are extra files available within the package, or if the "Index.xml" says that there are 1000 objects to be imported.

3. Once the XSLT transformation is complete, make the "Ready" file available to allow the content import jobs to actually import whatever is indicated by the modified "Content<nnn>.xml" files within the content package.

4. Key features

4.1. Mapping different metadata structures between vaults

For user-defined structures, it is possible to map the following metadata structures between different vaults: object types, value lists, property definitions, classes, workflows, workflow states, named ACLs, users, and user groups. The mapping order is as follows:

1. Semantic aliases (default: mapping enabled)
2. Local ID and name together (default: mapping enabled)
3. Name (default: mapping disabled)

The enabled/disabled status of the mapping order can be configured via registry settings (for more details, see the settings "MetadataStructureMappingWithAlias", "MetadataStructureMappingWithIdAndName" and "MetadataStructureMappingWithName").

Semantic aliases can be defined for each metadata structure element (except for users, for which the login account name is implicitly used as a semantic alias) in M-Files Server Administrator. For a successful mapping, there must exactly one match found. However, there can be multiple aliases (separated by a semicolon) defined for a single item. If the imported metadata structure element has multiple aliases, alias mapping is done one by one, and the first successful mapping is used.

Property definitions for owner properties are mapped automatically via their associated object types or value lists, thus there is no need to specify a semantic alias for those property definitions.

If the imported content package is lacking the alias information for some reason, there is an option ("Use the name of an imported element as its alias…") available for content import jobs to use the incoming names in place of missing aliases. Note that this setting affects the incoming data only; for a successful mapping, semantic aliases are still required in the replica vault.

Also note that built-in structures are always automatically mapped between vaults. For example, built-in properties such as "Created", "Last Modified", or "Comment" are mapped automatically, and no semantic aliases are needed for them. Similarly, built-in object types such as "Document", or built-in classes such as "Unclassified Document" are mapped automatically.

4.2. Object identification across multiple vaults

As a primary means of cross-vault identification, each object has a global identity based on its Object GUID assigned upon the object creation. Objects that originate from an external database have another strong identity with their external ID (in the context of the local object type or value list). If such an external object is created separately in multiple vaults, thus getting multiple different Object GUIDs, the object identity can be sufficiently determined by the external ID. The external ID typically represents the object identity in the external database, which means that the external identity is "global" from the M-Files point of view.

In addition to the global or external identities, each object has an original identity that is based on the original vault GUID (i.e., the vault in which the object was initially created), the original object type ID and

the original object ID. The original identity is typically used in file system level links embedded into documents. Additionally, the ID information shown in the M-Files client UIs is based on the original identity.

Finally, each object has a local identity comprising of a local object type ID and a local object ID. The local identity does not have any meaning outside of a single vault.

4.3. Mapping object identities across multiple vaults

Even if the global identity of objects does not directly match across multiple vaults, there are additional ways of mapping object identities. The following list describes how and in which order the object identity mapping is determined in various cases:

- Objects from real object types.
  1. Object GUID
  2. External ID (within the object type already mapped)
- Items from value lists that represent metadata structure elements (classes, workflows, workflow states, users, and user groups). Note that in this case, the Object GUID has been already resolved earlier via metadata structure mapping.
  1. Object GUID
- Items from value lists that are hierarchical. Note that name matching is not used here because the parent-child value hierarchy may contain items with equal names.
  1. Object GUID
  2. External ID (within the object type already mapped)
- Items from value lists that are subtypes of another type. Note that name matching is not used here because there may be multiple sub-items with the same name for different owner items. For example, multiple customers may have a contact person with the same name, say "Ann Jones". In such a case, the value list representing contact persons would contain multiple "Ann Jones" items, each one having a different customer as the owner value.
  1. Object GUID
  2. External ID (within the object type already mapped)
- Items from "plain" value lists that are not described above.
  1. Object GUID
  2. External ID (within the object type already mapped)
  3. Local ID and name together (within the object type already mapped)
  4. Name (within the object type already mapped)

4.4. Detecting conflicting changes across vaults

Conflict object is created when the incoming content package contains object versions that cannot "anchored" consistently with the locally existing versions. "Anchoring" is based on a globally unique identifier that is assigned to each object version when a new version is created.

For example, an object O initially has versions 1 and 2 in vault A, and it is replicated to vault B. Next, a new version 3 is created in parallel in both vaults A and B, and it is then replicated either way. In this case, the version 3 will be found in both vaults, but with a different global identifier, thus failing to "anchor" the version 3 properly. As a result, the incoming version 3 is imported into a conflict object. The conflict object always has a separate identity of its own, and a relationship is established from the conflict object to the object O (known as "base object"). A built-in property "Conflicts with" (specific to each object type) is created on demand for this purpose.

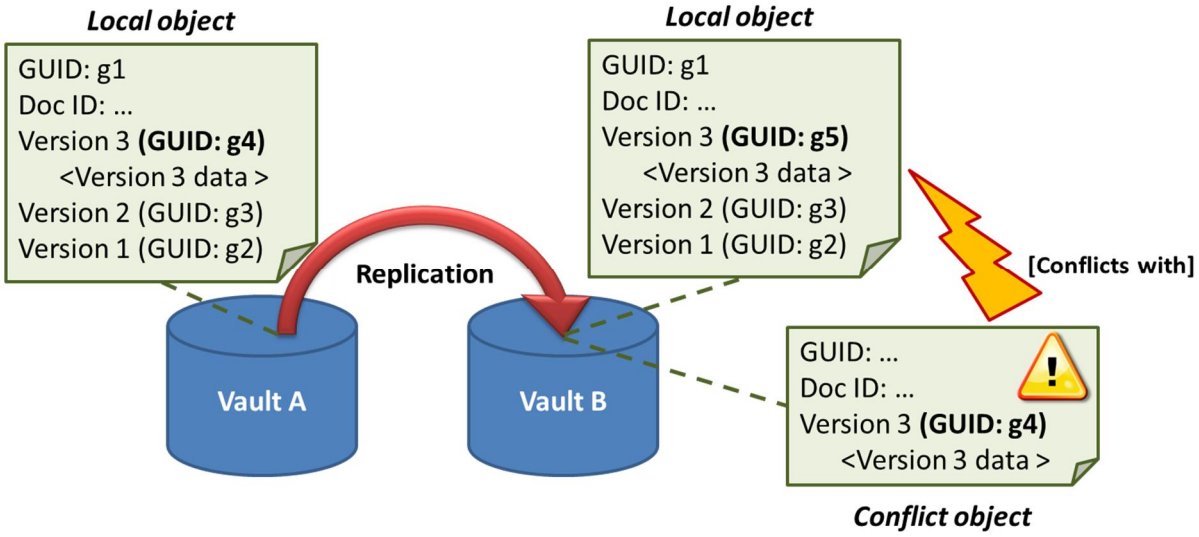

Fig. 12. Conflict is detected, and a conflict object is created.

As another example, an object O initially has versions 1 and 2 in vault A, and it is replicated to vault B. Next, a new version 3 is created in vault A. When this change is replicated to vault B, the version 3 is transferred with full data but for the versions 1 and 2 only the identity information is transferred. The incoming version 3 is obviously not found in vault B, but the continuous chain of older versions can be properly "anchored" at the version 2. This is not a conflict, and therefore no conflict object is created. However, the situation would be totally different if the versions 1 and 2 were destroyed from vault A before replication. Without having even identity information for the versions 1 and 2 in the content package, the "anchoring" cannot be done and a conflict object is created.

Conflict objects are specific to the original vault. This means that the object O may have multiple outstanding conflict objects, one for each original vault from which the conflicting changes have been imported. For a single conflict object, subsequent replication data is accumulated as long as it is imported from the same original vault.

4.5. Resolving conflicts

Once a conflict object has been created for a detected conflict, you can find the conflict object through related objects (see Fig. 4). If the object has conflicts, you can find them under the Conflicts grouping title. You can also find all conflict objects by means of the dedicated Conflicts view (hidden by default).

You can resolve conflicts by selecting the conflict object and by keeping the conflicting changes ("Keep These Changes"), or by discarding those ("Discard These Changes"), as appropriate. The former choice is about accepting the conflicting changes, whereas the latter choice results in local changes to be accepted. Conflict resolution involves all the conflicting versions, so there is currently no means of selectively accepting changes from both sides.

In order to resolve a conflict, you must have editing rights to the local object and the conflict object in the same vault (no access whatsoever is required to the conflicting object in the remote vault).

5. Frequently asked questions

5.1. Are there any additional requirements for a successful mapping of metadata structures?

Yes, the additional requirements are as follows:

- Object types and value lists
  - There must be no mismatch regarding real object type vs. pure value list
- Property definitions
  - Data types must match
  - For lookup properties, the referenced object types must be mapped accordingly
- Classes
  - Associated object types must be mapped accordingly
- Workflows and states
  - For each state, the associated workflow must be mapped accordingly (the mapped workflow-state pair must be valid also in the replica vault)

5.2. Does unmappable metadata survive replication roundtrips?

A replication roundtrip consists of two consecutive replications of the same data, one from the original vault to the replica vault, and another from the replica vault back to the original vault.

If the following metadata structure elements cannot be mapped at the replica end, associated metadata content cannot survive a replication roundtrip:

- Object types and value lists
- Property definitions
- Named ACLs (the named ACL link)
- Workflows and states (validated as a pair)

If the following metadata structure elements cannot be mapped at the replica, the actual metadata values for them survive the replication roundtrip anyway (the corresponding value list item reference is preserved):

- Classes
- Users
- User groups
- Named ACLs (the permissions content)

5.3. My replica server has a set of users that is completely different from that on the original server. How do I map the users and permissions in this case?

If a non-existent user is imported to a replica vault, a deleted value list item is created as a referenceable placeholder for that user. For such non-existent users, permissions referring to them are listed as "(deleted)", and property values referring to them include the "(deleted)" indicator in their display name.

Unfortunately, the current implementation does not support undeleting these items (or any other deleted value list items, for that matter).

With two-directional replication between vaults A and B, references to non-existent users in vault B will survive the whole roundtrip from A to B and back to A. This is due to the fact that the global identity of the items is preserved even when a "(deleted)" placeholder is created.

In order to avoid non-existent user to be created as deleted value list items, the following configuration options are available:

- Import job can be configured to use fixed permissions for imported objects, regardless of their imported permissions. Note that this does NOT affect any automatic permissions configured in the replica vault.
- Registry setting (see "ArchiveRestorePropertiesWithSpecificUserID") can be used to configure individual properties based on the Users value list to use a predefined item value. For example, the replica vault might have a special user "Publisher" to be used as a value for the "Created by" property.

5.4. Do M-Files URLs work across multiple vaults?

Yes, object URLs and file URLs referring across vaults can be resolved by the M-Files URL resolver. Other types of M-Files URLs do not work across multiple vaults.

URLs generated before M-Files 9.0 (without the "object" or "file" query string parameter)

The URL "m-files://show/ACF7FC8D-2A41-43FD-BACC-FDD895870357/0-91026" is resolved in M-Files 9.0 and later as follows:

1. If the vault connection to the indicated vault is available, the object identification is interpreted as local IDs within that vault. This is the default resolution step.
2. If the object was not found in step 1, all the available vault connections are then examined. A vault connection is considered available if the user is already logged in using the connection. For each such connection, the vault GUID and the object identification are interpreted as the original identity of the object (original vault GUID + original object type ID + original object ID) when searching.

Step 2 can be disabled by a registry setting, see "CanUseOriginalObjIDResolving" for more details.

URLs generated in M-Files 9.0 or later (with the "object" or "file" query string parameter)

The URL "m-files://show/ACF7FC8D-2A41-43FD-BACC-FDD895870357/0-91026?object=487B5435-7AC8-408E-BBB1-0EC206C7021E" is resolved in M-Files 9.0 and later as follows:

1. If the vault connection to the indicated vault is available, the object identification is interpreted as local IDs within that vault. This is the default resolution step.
2. If the object was not found in step 1, all the available vault connections are then examined. A vault connection is considered available if the user is already logged in using the connection. For each such connection, the value of the "object" query string parameter is used as an Object GUID when searching for the object.

The new URI format is more robust than the old one because of the explicit global object identity information. Also note that the new URI format is backward compatible: older M-Files clients are able to resolve new URIs by interpreting the local identity only.

Additionally, it is possible to optimize the vault scanning phase in step 2 by configuring preferenced vault connections to be used instead of scanning all available vault connections. See the registry settings "VaultPreferencesInURLMapping\Forced" and "VaultPreferencesInURLMapping" for more details.

5.5. Do embedded file links work across multiple vaults?

Embedded file links between two documents can be created in various applications. For example, Microsoft Word creates an embedded file link when a Microsoft Excel spreadsheet is copied and pasted into a Word document. In a generic case, embedded file links work if the links have been established when the files at the both ends of the link already exist in M-Files, AND if the links are relative.

Typically, external applications access the files in M-Files 9.0 and later via the ID2 view that is based on the original identity. The original identity is maintained for the replicated objects, and it consists of the original vault GUID, the original object type ID, and the original object ID.

Also note that similarly to older versions of M-Files, the M-Files AutoCAD Add-in has more intelligence for adjusting also the absolute file links between drawings.

5.6. My new value list item appears as "My New Item (deleted)" in other vaults, why?

Normally during the import, a missing value list item (referred to by a property value) would be created in the target vault as if it was created locally. However, the missing value list item is created as *deleted* in the following cases:

- Item belongs to a value list that represents metadata structure elements (classes, workflows, workflow states, users, and user groups)
- Item belongs to a value list that is hierarchical
- Item belongs to a value list that is a subtype of another type

5.7. How do I find objects that have property values referring to deleted items?

1. Go to Advanced Search
2. Select a lookup property you are interested in
3. In the "Specify value" combo box, select the "Filter: …" item
4. Specify "*" as the filter, and check the "Include deleted items" option

5.8. Is it possible to modify the metadata structure once the replication has started?

Yes, but any modifications to the structure must be imported to each vault participating in the replication.

As a rule of thumb, metadata structure changes should be exported from a single master vault and then imported to other (slave) vaults. Please note that removals of metadata structure elements are not currently replicated.

5.9. I receive e-mail notifications from multiple vaults for the same object, why?

In each vault, notification rules are applied based on the local M-Files event log data. When replicated data is imported to a replica vault, event log entries are written in the same way as if the objects are created or modified locally in that vault. As a result, notifications are generated from the original data and the replicated data alike, thus effectively sending multiple notifications for a single object operation, one for each vault participating in replication.

As a workaround, apply the following steps in replica vaults:

- Disable built-in notification rules for assignments
  (via registry settings "BuiltInAssignedToRuleEnabled" and " BuiltInCompletedByRuleEnabled" under HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer)
- Add the corresponding rules for assignments as common notification rules
- For individual users, disable their notification rules in other vaults but their original vault

Also note that notifications for the imported data are currently sent from the "M-Files Server" user identity, not from the actual user who is responsible for the changes of the original data.

It is probable that there will be improvements in this area in future versions of M-Files.

5.10. What happens if the local object has newer changes than those being imported from a content package?

For versioning changes, and if the local object has newer versions than the incoming object, the outcome depends on the incoming versions. If the incoming object versions have clashing identities (version number plus a globally unique identifier) with the local object versions, a conflict object is created. If the version identities do not clash and the version collection of the incoming object is a genuine subset of the version collection of the local object, the incoming object is simply skipped.

For non-versioning changes, a newer change (indicated by the corresponding change timestamp) always wins. Any incoming changes that are older than the local changes are simply skipped.

5.11. Some objects in the original vault are not exported, why?

1. Objects with external files are not exported.
2. The latest version of the object does not match the export filter if specified.
3. In delta export, the value of the ObjectChanged property is not newer than the delta timestamp of the export job.
4. If the object has been destroyed, the destruction information must be exported separately.

In some cases, there is further information (IDs of the skipped objects, for instance) available from the Windows event log.

5.12. Some objects in the content package are not imported, why?

1. The object type of an incoming object cannot be successfully mapped in the target vault:

a. object type cannot be found at all

b. real object type is mapped to a pure value list, or vice versa

c. object already exists with a different object type

2. The mapped object type is external in the target vault. In the current implementation, no changes are imported for external objects, even if the changes are limited to custom properties (i.e., those properties that do not have counterparts in an external data source).

3. The incoming object ACL definition cannot be mapped in the target vault, AND there is no overriding ACL provided. The incoming object ACL definition cannot be mapped if there are pseudo-user references using object types or property definitions that cannot be mapped.

4. The original identity of an incoming object clashes with the existing original identities in the target vault. This is a possible scenario when the target vault has been restored from a backup, and objects originating from the target vault have not been imported from other vaults before creating new objects in the target vault.

5. The incoming object is originally a conflict object. Conflict objects are only created from the conflicting changes.

6. An existing conflict object has been locally modified (should not be possible anyway) and is about to cause a conflict on a conflict. No chain of conflict objects is constructed.

7. A shortcut object must not overwrite an existing full object.

8. A destroyed shortcut object must not cause a full object to be destroyed.

9. The current checkout status of a shortcut object is not overwritten. If there is full version data available for the incoming object, the shortcut object is normally promoted to a full object.

In some cases, there is further information (IDs of the skipped objects, for instance) available from the Windows event log.

5.13. Some object data in the content package is not imported, why?

1. Property values are not imported if their property definition cannot be mapped.

2. The properties "Accessed by me" and "Favorite views" are not replicated.

3. The "Last accessed" value of the files is not imported if there is no other change on the file.

5.14. When an import job is run in my vault, the timestamps indicating the latest run of export jobs seem to be adjusted backwards. Why?

When the import option "Reset export timestamps" is turned on, the following additional steps takes place:

- During the import, the oldest relevant change caused by the import job is tracked. The oldest relevant change is determined by the change timestamp for the oldest change that is actually imported to the target vault.

- After the import, the delta timestamp of each export job on the target server is updated based on the oldest relevant imported change.

- As a result, it is possible to automatically export recently imported changes in a daisy-chain fashion when a delta option is applied.

Note that in some cases this feature may cause suboptimal results. For example, when an export filter is modified, the new export set may include previously unseen objects that have arbitrarily old timestamps. On the import side, this may lead to very large export sets due to a wide delta range.

5.15. I have configured my export job to save files as PDF/A. Why cannot I find them in any remote vault after import?

The main purpose of exporting PDF/A files is long-term archiving. Exported PDF/A files are saved to the content package, but they are not imported along with the rest of the data. Within the content package, you can locate the PDF/A files by looking for "<pdfa>" elements in the content XML files.

5.16. Is it possible to drop IDs from the names of the files in the content package?

Yes, there is a registry setting available for this. See "ArchiveIncludeIDsInNamesInFileStructure" in Appendix A.

5.17. Non-admin users are not allowed to use the "Mark for archiving" command any longer. How do I make that possible again?

The default permissions for the "Marked for archiving" property deny the edit access for everybody. However, those permissions can be adjusted in M-Files Server Administrator:

1. Go to the vault
2. Go to "Metadata Structure (Flat View)"
3. Go to "Property Definitions"
4. Click "Show All Property Definitions"
5. Select "Marked for archiving" and go to properties
6. Go to the "Permissions" tab
7. Adjust the permissions and click OK

5.18. What exactly is stored into a conflict object?

A conflict object contains all the version data for its base object at and above the conflict level. The conflict level designates the version at which the conflict originally occurred. Once the conflict object has been created, subsequently incoming changes address either the conflict object or its base object, depending on whether they affect the versions above or below the conflict level. Object-specific changes (deletion, checkout, etc.) always address the base object.

5.19. What am I supposed to do with conflict objects?

An existing conflict object always indicates that there are mismatching modifications between two vaults. Therefore, any outstanding conflict should be resolved before continuing to use the object normally. In conflict resolution, all the conflicting changes for a single object can be either kept or discarded. After conflict resolution, the conflict object will remain as deleted, now containing the discarded changes. The

conflict resolution can be first undone by undeleting the resolved conflict object, and then reversed by resolving the conflict again by keeping the conflicting changes.

Note that upon resolving, the current implementation assigns a new identity to each document file in case of keeping the conflicting changes. This means that the files of the conflicting versions will get new file IDs (local identity) and GUIDs (global identity).

5.20. I am replicating my object between two vaults, and now I have a conflict object at both ends. How do I resolve this conflict correctly?

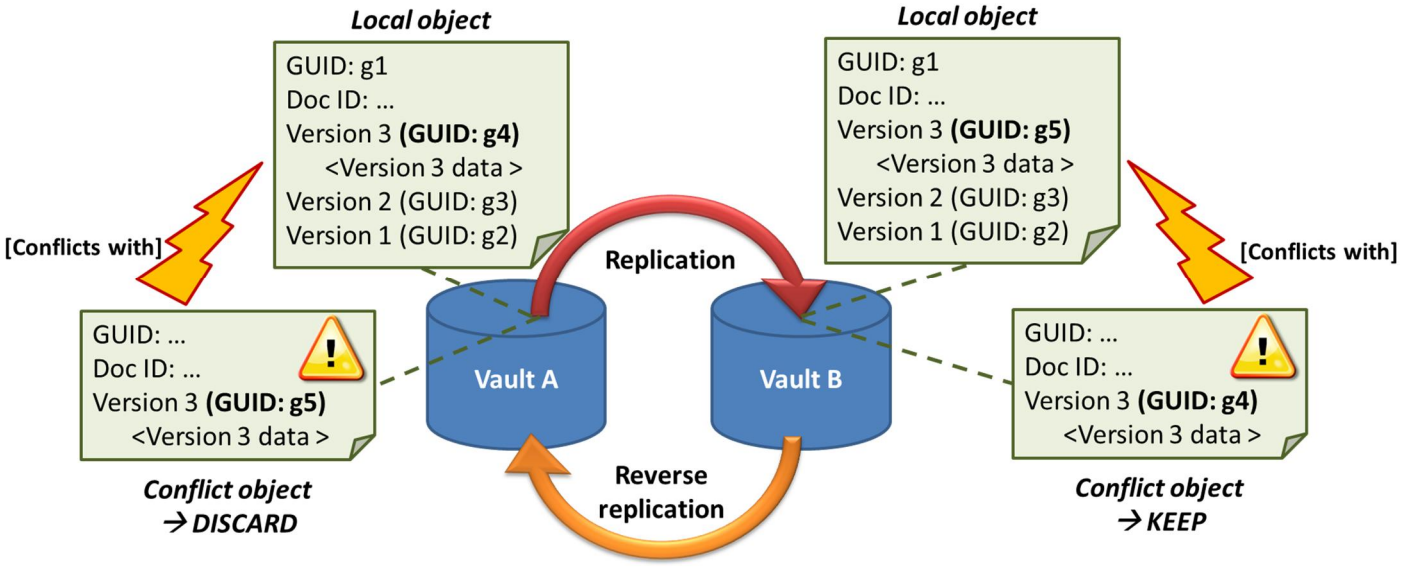

Fig. 13. Resolving a conflict in two-directional replication.

A conflict object is always created from the incoming data. Therefore, the resulting conflict objects represent opposite changes which must be resolved correspondingly. If you want to keep the changes originally made in Vault 1, we need to discard the conflicting changes in Vault 1, whereas you need to keep the conflicting changes in Vault 2.

5.21. What are the timestamp properties relevant to this feature?

There are several new timestamp properties that are used to track various changes on objects, and to control whether those changes are to be included in the delta export.

- Object level
    - OBJECTCHANGED (new) indicates the latest overall change on the object. This timestamp is updated when any of the other timestamp values is updated.
    - DELETESTATUSCHANGED (new) indicates the latest time when the object was deleted or undeleted.
    - CONFLICTRESOLVED (new) indicates the latest time when the conflict was resolved on this object.
- Object version level

- STATUSCHANGED indicates the latest time when the checkout/checkin state was changed on this object version.
- ACLCHANGED (new) indicates the latest time when the permissions were modified on this object version.
- VERSIONLABELCHANGED (new) indicates the latest time when the version label was added, modified, or removed on this object version.
- VERSIONCOMMENTCHANGED (new) indicates the latest time when the version comment was added, modified, or removed on this object version.

5.22. Can I use metadata-driven permissions normally via shortcut objects?

No, because property data on a shortcut object is very limited; typically the name property is the only relevant property value on shortcut objects. This means that shortcut objects do not contribute to the pseudo-user resolution in object permissions: property values are retrieved from the local data only, which means that any pseudo-user references via a shortcut object remain unspecified. As an example for a workaround, a document that specifies permissions for "Project.ProjectManager" must always be replicated together with the project it refers to.

5.23. Can I use any event handlers with replication and archiving?

There are two event handlers available (it is not possible to modify the imported data):

- "Replication: AfterCreateNewObjectFinalize" is triggered when a new object is created upon content import.
- "Replication: AfterCheckInChanges" is triggered when an existing object is changed upon content import.

Appendix A. Common registry settings

Export

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	ArchiveIncludeIDsInNamesInFileStructure
Data Type	REG_DWORD
Value Data	0 -> disabled 1 -> enabled (default)
Description	If disabled, the file names in the content package do not include the object ID. This setting is provided as a counterpart for the client-side registry setting IncludeIDsInNamesInIDView.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	ArchiveAvoidDuplicateDocumentFilesInIDStructure
Data Type	REG_DWORD
Value Data	0 -> disabled 1 -> enabled (default)
Description	If enabled, the document files are stored only once per file version in the content package structure. Because the latest version is always stored anyway, the older file versions are stored with the object version that is the latest one before the next file version change. In the following simplified example, the file versions that are actually to be stored are marked with "*" : • object version 5, file version 3 * • object version 4, file version 2 * • object version 3, file version 2 • object version 2, file version 1 * • object version 1, file version 1 If disabled, the full set of document files for each object version is stored as such.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer\Vaults\<vault>**
Value name	ArchiveExcludeProperties
Data Type	REG_SZ_MULTI
Value Data	Local property definition IDs (one for each line of the multi-string)
Description	The specified properties are excluded from the exported data. Note that only some of the built-in properties can be excluded (see PropertyDefs.h for details).

Key	HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer
Value name	ArchiveBypassMaximumFileSizeCheckInPDFAConversion
Data Type	REG_DWORD
Value Data	0 -> false 1 -> true (default)
Description	If true, normal size limits configured for the PDF conversion are not applied when used from replication and archiving.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	ArchivePDFConversionTimeoutInSeconds
Data Type	REG_DWORD
Value Data	Timeout value in seconds (default: 120)
Description	Timeout value to give up PDF/A conversion on a single file if conversion takes longer than that.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	ArchiveOldVersionsBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 1000)
Description	Batch size for exporting old object versions. Each batch is run as a separate database transaction.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	ArchiveObjectsBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 1000)
Description	Batch size for exporting object data. Each batch is run as a separate database transaction.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	DestroyOldVersionsUponArchivingBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 100)
Description	Batch size for destroying old object versions after exporting them. Each batch is run as a separate database transaction.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	DestroyObjectsUponArchivingBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 100)
Description	Batch size for destroying objects after exporting them. Each batch is run as a separate database transaction.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	ResetMFAUponArchivingBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 100)
Description	Batch size for resetting the "Marked for archiving" property value on objects after exporting them. Each batch is run as a separate database transaction.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	LogUponArchivingBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 100)
Description	Batch size for writing export log entries about objects after exporting them. Each batch is run as a separate database transaction.

Import

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer\Vaults\<vault>**
Value name	ArchiveExcludePropertiesUponRestore
Data Type	REG_SZ_MULTI
Value Data	Local property definition IDs (one for each line of the multi-string)
Description	The specified properties are excluded from the imported data. Note that only some of the built-in properties can be excluded (see PropertyDefs.h for details).

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer\Vaults\<vault>\ArchiveRestorePropertiesWithSpecificUserID**
Value name	<local property definition ID indicating the property>
Data Type	REG_SZ
Value Data	Local user ID as a string value.
Description	The specified properties are imported with the specified user ID value, regardless of the imported data. The specified property definition is ignored if it is not based on the Users list.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer\Vaults\<vault>**
Value name	MetadataStructureMappingWithAlias
Data Type	REG_DWORD
Value Data	0 -> disabled 1 -> enabled (default)
Description	If enabled, metadata structure elements can be mapped by using their semantic aliases.

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer\Vaults\<vault>**
Value name	MetadataStructureMappingWithIDAndName
Data Type	REG_DWORD
Value Data	0 -> disabled 1 -> enabled (default)
Description	If enabled, metadata structure elements can be mapped by using their ID and name together (name in vault default language).

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer\Vaults\<vault>**
Value name	MetadataStructureMappingWithName
Data Type	REG_DWORD
Value Data	0 -> disabled (default) 1 -> enabled
Description	If enabled, metadata structure elements can be mapped by using their names only (in vault default language).

Key	**HKEY_LOCAL_MACHINE\SOFTWARE\Motive\M-Files\<version>\Server\MFServer**
Value name	RestoreFromArchiveBatchSize
Data Type	REG_DWORD
Value Data	Number of objects processed within a single batch (default: 100)
Description	Batch size for importing objects. Each batch is run as a separate database transaction.

URL handling

Key	**HKEY_CURRENT_USER\SOFTWARE\Motive\M-Files\<version>\Client\MFStatus\VaultPreferencesInURLMapping\Forced**
Value name	<vault GUID as in URL>
Data Type	REG_SZ
Value Data	Vault GUID that is used instead of the GUID in the URL.
Description	Only the vault indicated by the value of this setting is accessed when trying to resolve the URL target. If the user is not logged in the vault, the login prompt is shown normally.

Key	**HKEY_CURRENT_USER\SOFTWARE\Motive\M-Files\<version>\Client\MFStatus\VaultPreferencesInURLMapping**
Value name	<vault GUID as in URL>
Data Type	REG_SZ
Value Data	Vault GUID that is used instead of the GUID in the URL.
Description	First, the vault GUID in the URL is used when trying to resolve the URL target. If the URL cannot be resolved, the vault indicated by the value of this setting is accessed next. If the user is not logged in that vault, the login prompt is shown normally. If the URL cannot still be resolved, rest of the available (i.e., the user is logged in) vaults are accessed.

Key	**HKEY_CURRENT_USER\SOFTWARE\Motive\M-Files\<version>\Client\MFStatus**
Value name	CanUseOriginalObjIDResolving
Data Type	REG_DWORD
Value Data	0 -> disabled 1 -> enabled (default)
Description	If enabled, the object identity within the old M-Files URLs (created before M-Files 9.0) can be interpreted as the original object identity (original vault GUID, original object type ID, original object ID).

Appendix C. XML processing

Using XSL transformation to select which objects to import

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:param name="objecttype" select="objecttype"/>
   <xsl:param name="objectid" select="objectid"/>
   <xsl:param name="latesttitle" select="latesttitle"/>
   <xsl:template match="@*|node()">
      <xsl:copy>
         <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
   </xsl:template>
   <xsl:template match="/content/object">
      <xsl:choose>
         <!-- Should we use the object ID filter? -->
         <xsl:when test="string-length($objecttype) > 0 and string-length($objectid) > 0">
            <xsl:if test="objid[@objtype=$objecttype][@id=$objectid]" >
               <xsl:copy>
                  <xsl:apply-templates select="@*|node()"/>
               </xsl:copy>
            </xsl:if>
         </xsl:when>
         <!-- Should we use the latest title filter? -->
         <xsl:when test="string-length($latesttitle) > 0">
            <xsl:if test="version[@lci='true']/properties/property[@id='0']=$latesttitle" >
               <xsl:copy>
                  <xsl:apply-templates select="@*|node()"/>
               </xsl:copy>
            </xsl:if>
         </xsl:when>
         <xsl:otherwise />
      </xsl:choose>
   </xsl:template>
</xsl:transform>
```

Content<nnn>.xml files can be modified to contain only those objects that actually need to be imported. The XSL script above shows an example how to filter the desired objects based on either their latest title or object identification.

To apply this transformation, a command line tool "msxsl.exe" is available from Microsoft Download Center. The following command line examples illustrate how to overwrite existing Content<nnn>.xml files with filtered ones:

```
msxsl Content1.xml Filter.xsl -o Content1.xml latesttitle='General Meeting 1/2004'

msxsl Content1.xml Filter.xsl -o Content1.xml objecttype=0 objectid=123

for %i in (Content*.xml) do msxsl %~ni.xml Filter.xsl -o %~ni.xml latesttitle='General
Meeting 1/2004'
```